2016

C++

principles of object oriented programming

C++ is an enhanced version of the C language. C++ includes everything that is part of C language and adds support for object oriented programming (OOP). With very few, very major exceptions, C++ is a superset of C.

Rome
Sea nest
7/5/2016

WELCOME

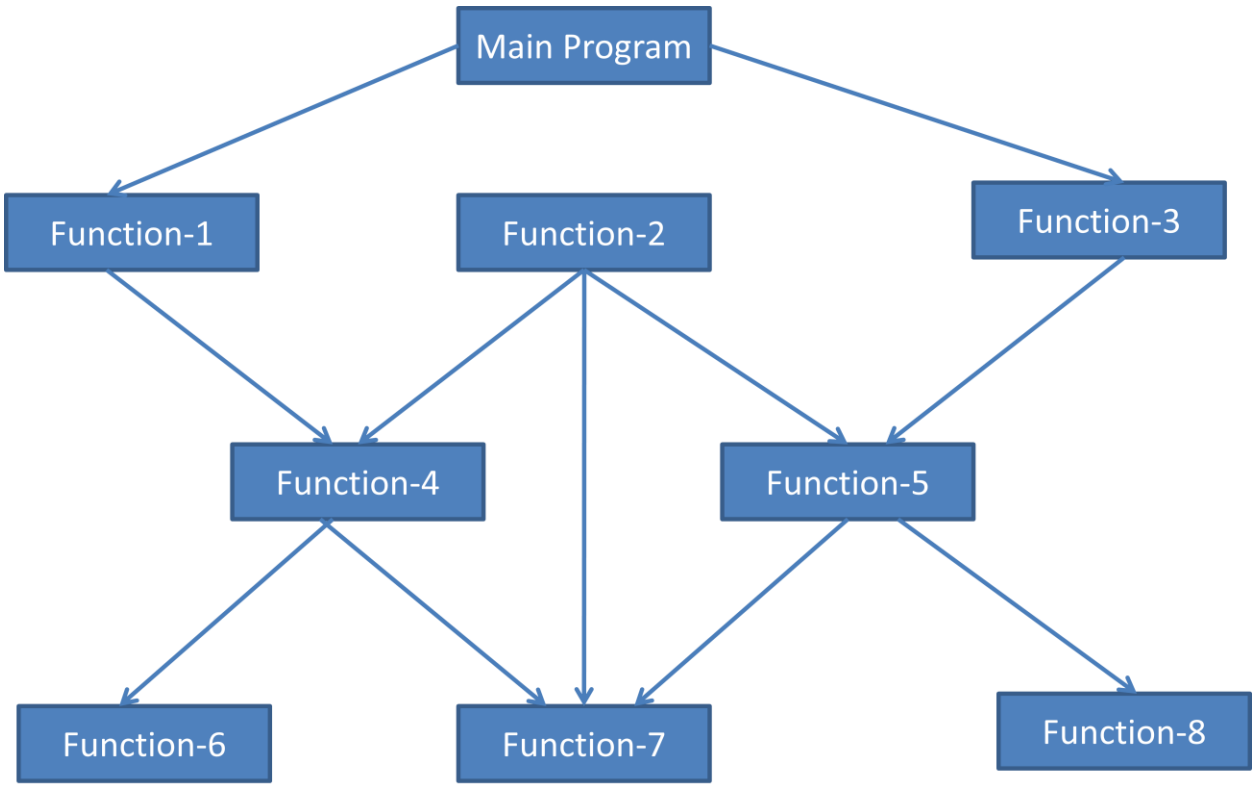
principles of object oriented programming

Concept about C++:

C++ is an enhanced version of the C language. C++ includes everything that is part of C language and adds support for object oriented programming (OOP). With very few, very major exceptions, C++ is a superset of C.

A look at procedure oriented programming:

Conventional programming, using high level languages such as COBOL, FORTRAN and C programming languages , is commonly known as procedure-oriented programming (POP). In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions. A typical program
structure for procedural programming is shown in Fig. 1.4.



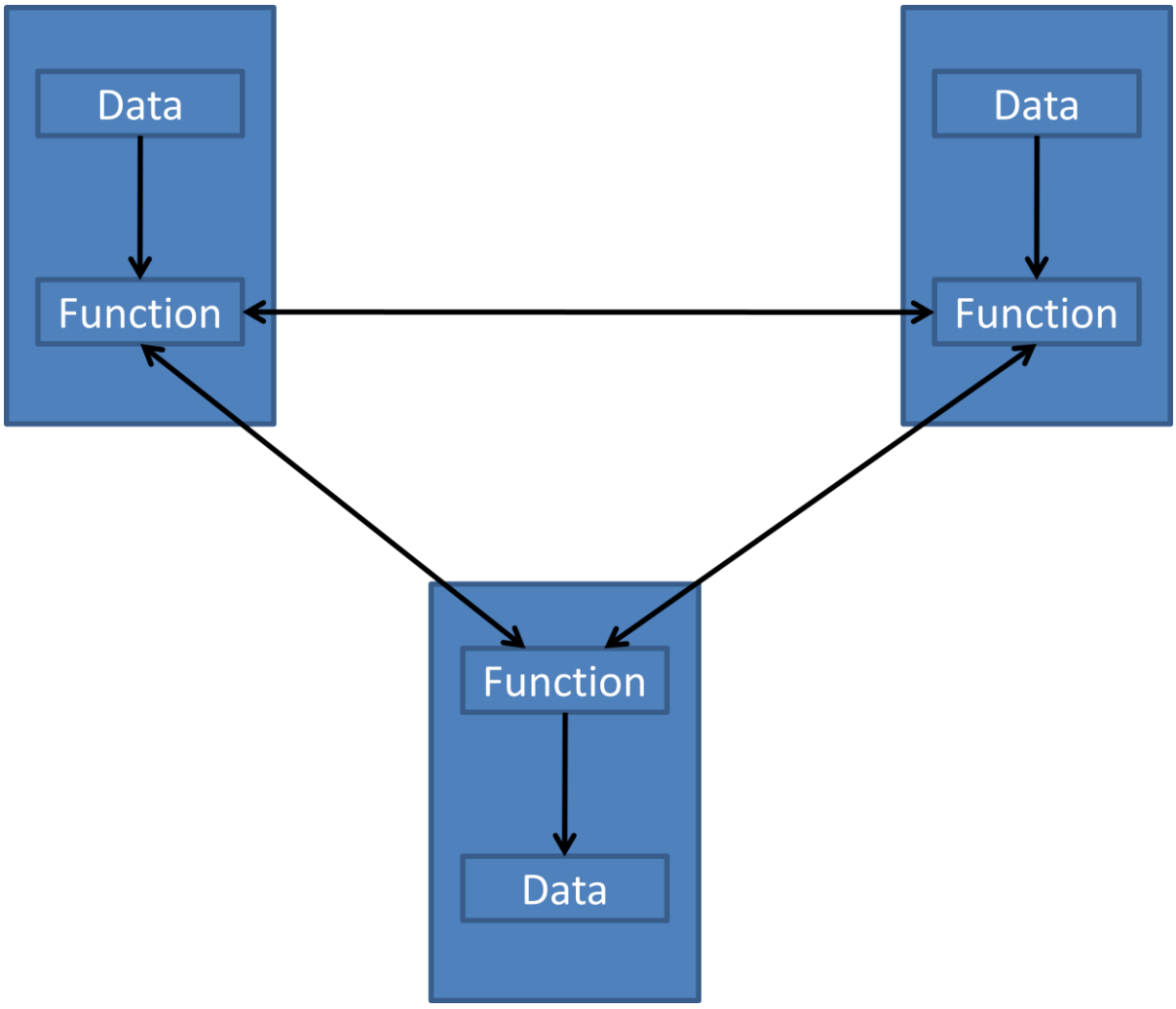
Main characteristics of procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.

- Employs top-down approach in program design. Top down and bottom up are designing approaches. In top down approach, first we are designing the main module (i.e main function) and then we will design all other sub modules.

Object-Oriented Programming paradigm:

The major motivating factor in the invention of object-oriented approach is to remove some of the flows encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called object and then builds data and functions around these objects. The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.



Striking features of object-oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.

- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follow bottom-up approach in program design. In bottom up approach, first we design all the sub modules related to application then we design main module.

C++ Console I/O:

In C programming language, we are using printf() function for output operation and scanf() function for input operation. But in C++, cout<< is using for output operation and cin>> function is using for input operation. "<<" operator is called insertion or put to operator . ">>" operator is called extraction or get from operator.

Example:

```
cout<<"Object-Oriented Programming ";
cout<<100.99;

int num;
cin>>num;
```

Program:

```
1) #include<iostream.h>
//C++ console i/o operator
int main()
{
int i, j;
double d;
i=10;
j=20;
d= 99.101;
cout<<"Here are some values:";
cout<<i;
cout<<' ';
cout<<j;
cout<<' ';
cout<<d;
```

```
return 0;
}
```

Output:
Here are some values- 10 20 99.101

Assignment:
Write a program that uses C++ style I/O and produce the output as:
6.62 9.34 99 110

```
#include<iostream.h>

int main()
{
int i;
float f;
char s1[40], s2[40], s3[40];
cout<<"Enter an integer, float and string:";
cin>>i>>f>>s1>>s2>>s3;
cout<<"Here's your data:";
cout<<i<<' '<<f<<' '<<s1<<' '<<s2<<' '<<s3;
return 0;
}
```

C++ Comments

In C++, we can include comments in our program two different ways such as-

1. For multiline comments, began a comment with /* and end it with */
2. For single line comments, began a comment with // and stop at the end of the line.

Program:

```
3)#include<iostream.h>
//Comments program
int main()
{
char ch;   //this is a single line comment
/* this is a
multi line comment*/
```

```
cout<<"Enter keys, x to stop.\n";
do
{
cout<<":";
cin>>ch;
}while(ch!='x');
return 0;
}
output:
enter keys, x to stop:
dgsdds sdsd
x
```

Classes and Objects

Limitation of C structures and benefits of C++ class:

1) Suppose student is a structure name in C language can be define as

```
struct student
{
 Int roll_number;
float marks;
};

main()
{
```

```
struct  student C1,C2,C3; // C struct type variable declaration.

C3=C1+C2; // it is not possible to add two string type variable in C programming language.
}
```

it is not possible to add two structure type variable in C programming language. But in C++, we can add two class type objects like data type variable as shown bellow. This facility in C++ called **Operator Overloading.**

```
class semp
{
 Int roll_number;
float marks;
-------
-------
};

main()
{
semp  A1,A2,A3;  // class type object declaration.

A3=A1+A2; // it is possible to add two object in C++ language.
}
```

2) C++ class provides data hiding facility. C++ class can declare some of its members as **private** so that they cannot be accessed directly by the external functions. This is called data hiding facility.But in C structure, datas are declared only as **public** by default. So C structure cannot provide data hiding facility.

3) C++ support inheritance facility. Inheritance is a mechanism by which one type can inherit characteristics from other types, is also supported by C++.

4) In C, we need to use struct keywords for structure type variable declaration. But in C++, we no need to use class keywords for class type object declaration.

Object:

In the real world, Objects are the entities of which the world is comprised. Everything that happens in the world is related to the interactions between the objects in the world. Just as atoms, which are objects, combine to form larger objects, the interacting entities in the world can be thought of as interactions between and among both singular ("atomic") as well as compound ("composed") objects. The real world consists of many, many objects interacting in many ways. While each object may not be overly complex, their interactions create the overall complexity of the natural world. It is this complexity that we wish to capture in our software systems. In an object-oriented software system, objects are entities

used to represent or model a particular piece of the system. Objects are the primary units used to create abstract models.

An object that does not interact with anything else effectively does not exist. Access to internally stored data is necessarily through some sort of defined behavior of the object. It is impossible for an outside entity to truly "know" whether or not a particular piece of data is being stored inside of another object. The real meaning of the object is in how the object behaves in relationship to its data, should it contain any. The existence of data in an object is an implementation technique used to generate the required behavior of that object.

Objects are variables of type class. Once a class has been defined, we can create any number of objects belonging to that class. Objects contain data and code to manipulate that data.

Classes :

Many objects differ from each other only in the value of the data that they hold. For example, both a red light and a blue light are lights; they differ only in the value of the color attribute, one has a red color and the other a blue color. Our object-oriented system needs a way to capture the abstraction of a light, independent of the value of its color. That is, we want to express that a set of objects are abstractly equivalent, differing only in the values of their attributes. Object-oriented systems use classes to express the above concept of abstract equivalence. A class is a summary description of a set of objects. A class thus contains the descriptions of all the behaviors of the objects that it represents. In computer science parlance, we call the individual behaviors of a class its methods. In addition, a class contains descriptions of the internal data held by the objects. Turning the description around, we can say that a class is a pattern for the creation of a particular type of object. That is, one can use a class to create objects of the type described by the class. A class is an arrangement of a set of objects, it is not the actual object. In technical terms, a class defines a new type in the system. Types are identifies used to differentiate different kinds of data. For instance, integers, characters, strings and arrays are all types of data.

A class is a way to bind the data and its associated functions together. It allows the data (and function) to be hidden, if necessary from external use. A class is a collection of objects of similar type. Classes are user defined data types and behave like the built in data types of a programming language.

```
example:
class muna
   {
     memberfunction()
      {
      datatype:
      }
    };
main()
  {
   muna ob1;
   ob1.memberfunction();
   }
```

Example: if **a** is a class name and **ob** is a object of class a, then the format of using class and object in C++ program is look like as follows:

```
class a
{
public:
int x;
void getdata()
{
}
void display()
{
}
};

 void main()
{
a ob; //object declaration
ob.getdata();// accessing class member function
ob.display(); //accessing class member function
}
```

Specifying a class:

Generally, a class specification has two parts:

1. Class declaration
2. Class function definition

The class declaration describes the type and scope of its members. The class function definitions describe how the class functions are implemented.

The general form of a class declaration is:

```
class class_name
{
private:
 variable declarations;
function declarations;

public:
variable declarations;
function declarations;
};
```

The body of a class is enclosed within braces and terminated by a semicolon. The class body contains the declaration of variables and functions. These functions and variables are collectively called class members. Class members are usually grouped into two sections, namely, private and public. The keywords private and public are known as visibility labels. Note that these keywords are followed by a colon.

The class members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also. The data hiding (using private declaration) is the key features of object oriented programming. The use of the keyword is optional. By default the members of a class are private.

The variables inside the class are known as data members and functions are known as member functions. Only the member functions can have access to the private data members and private functions. However, the public members (both function and data) can be accessed from outside the class.

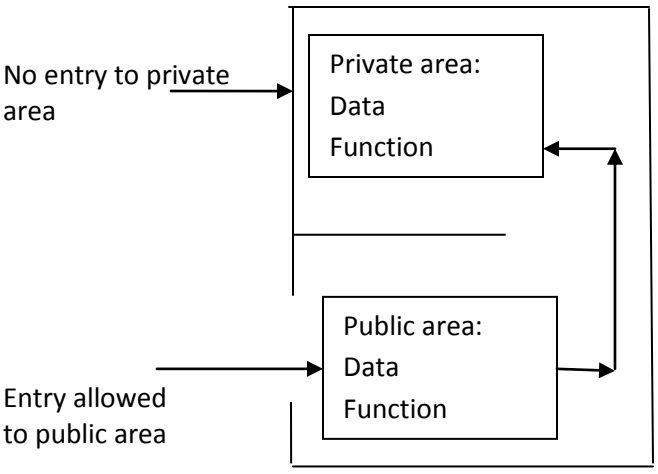


Figure: Data hiding in a class

Accessing class members:
We can access class members like the following way-
Syntax:
object_name.function_name(argument_list);

example:
ob.getdata();
ob.display();

Defining member function:
Member functions can be defined in two places:

- Outside the class definition
- Inside the class definition

Outside the class definition:
Member functions that are declared inside a class have to define separately outside the class.
return_type cass_name :: function_name (argument declaration)
{
Function body
}

Example:
void a::getdata(int a, float b)
{
number=a;
cost=b;
}

The symbol :: is called scope resolution operator.

The member functions have some special characteristics that are often used in the program development. These characteristics are:

- Several different classes can use the same function name.
- Member functions can access the private data of the class. A non member function cannot do so. (However, an exception to this rule is a fiend function discussed later.)
- A member function can call another member function directly, without using the dot operator.

Inside the class definition:
We can define member functions inside the class as well. Only small functions are defined inside the class.
class ae
{
public:
int x;
void getdata()

```
{
Int x=10;
}
void display()
{
cout<<x;
}
};

 void main()
{
a ob; //object declaration
ob.getdata();// accessing class member function
ob.display(); //accessing class member function
}
```

What are the advantages of OOP?

It presents a simple, clear and easy to maintain structure.

It enhances program modularity (partitioning) since each object exists independently.

New features can be easily added without disturbing the existing one.

Objects can be reused in other program.

Data is fully secured

Operator overloading is possible in OOP

Various elements of OOP are:

- Object
- Class
- Method
- Encapsulation
- Information Hiding

- Inheritance
- Polymorphism

Class:-

A class is a collection of data and the various operations that can be performed on that data.

Object-

This is the basic unit that are associated with the data and methods of a class. To access any element of a class, an object is used.

Method:-

A set of programming code used to perform the operation of the program. For example, find_salary().

Encapsulation:

Encapsulation is the process of binding class elements (data and functions) in a single unit. Encapsulation separates the conceptual interface. Encapsulation is the process of hiding data of a class from objects. This is to say, if a function or a method inside this class is private, only objects of this class can access the method. The three access specifiers (private, public, protected) determine whether the methods are accessible to all classes or only the defining class or the friend class.

Information Hiding:

Information hiding is the primary criteria of system and should be concerned with hiding the critical element in system. Information hiding separates the end users from illegal access of the requirement. **Encapsulation** supports information hiding by making use of the three access specifiers of a class.

Public:- If a class member is public, it can be used anywhere without the access restrictions.

Private: if a class member is private, it can be used only by the members and friends of class.

Protected : if a class member is protected, it can be used only by the members of class, friends of class and derived class.

Inheritance:

Inheritance is a way by which a subclass inherits attributes and behaviors of a parent or base class and can have its own as well. Inheritance is the process of involving building classes upon existing classes. So that additional functionality can be added. Using inheritance the hierarchical relationships are established. Inheritance allows the reusability of an existing functions and data of a class.

Polymorphism:

Polymorphism is the ability to calls the same named methods from different classes. Each function contains its own specific behavior. Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances.

Program:

```
4) #include<iostream.h>
//accessing private data using member function

class myclass
{

int a;
public:
void set_a(int num);
int get_a();
};
```

```
void myclass::set_a(int num)
{
a=num;
}

int myclass::get_a()
{
return a;
}

int main()
{
myclass ob1, ob2;
ob1.set_a(10);
ob2.set_a(99);

cout<<ob1.get_a()<<"\n";
cout<<ob2.get_a()<<"\n";
return 0;
}

output: 10
        99


5) #include<iostream.h>
//accessing public data using objects

class myclass
{
public:
int a;

};


int main()
{
myclass ob1, ob2;
ob1.a=10;
ob2.a=99;

cout<<ob1.a<<"\n";
cout<<ob2.a<<"\n";
return 0;
}

output: 10
```

Program:

```
 // test_circle.cpp
#include <iostream.h>


class Circle
{
  public:
    Circle(double X, double Y, double R);  // a constructor
    void Show();                          // a member function
    void Set(double R);                   // change the radius
    void Move(double X, double Y);        // move the circle
  private:
    double Xcenter;
    double Ycenter;
    double radius;
};



Circle::Circle(double X, double Y, double R)
{
  Xcenter = X;
  Ycenter = Y;
  radius  = R;
}

void Circle::Show()
{
  cout << "X, Y, R " << Xcenter << "  " << Ycenter << "  "
       << radius << endl;
}

void Circle::Set(double R)
{
  radius = R;
}

void Circle::Move(double X, double Y)
{
  Xcenter = X;
  Ycenter = Y;
}
```

```
int main()
{
 Circle c1(1.0, 2.0, 0.5);  // construct an object named 'c1' of type 'Circle'
 Circle circle2(2.5, 3.0, 0.1);  // another object named 'circle2'

 c1.Show();      // tell the object c1 to execute the member function Show
 circle2.Show();  // circle2 runs its member function Show

 c1.Move(1.1, 2.1);  // move center
 c1.Show();

 circle2.Set(0.2);   // set a new radius
 circle2.Show();
 return 0;
}
```

```
Output:
X, Y, R 1   2   0.5
X, Y, R 2.5   3   0.1
X, Y, R 1.1   2.1   0.5
X, Y, R 2.5   3   0.2
```

Private Section

- All data at the beginning of a class defaults to private and the private keyword is optional. This means, the data at the beginning of the class cannot be accessed from outside of the class; it is hidden from any outside access.
- Therefore, the variable named keep_data which is part of the object named John_cat defined in line 37 and 38, is not available for use anywhere in the main() program. That is why we have to comment out the following codes:

  // John_cat.keep_data = 100;
  // Joe_cat.keep_data = 110;
- It is as if we have built a wall around the variables to protect them from accidental corruption by outside programming influences.

Public Section

- A new keyword public, introduced in line 10 which states that anything following this keyword can be accessed from outside of this class as shown below:

```
public:   // public part
```

- Because the two functions are declared following the keyword public, they are both public and available for use by any calling program that is within the scope of this object.
- This essentially opens two small peepholes in the solid wall of protection that we built around the class and the private keep_data variable is not available to the calling program.
- Thus, we can only use the variable by calling one of the two functions defined within the public part of the class. These are called member functions because they are members of the class.

Since we have two functions, we need to define them by saying what each function will actually do. This is done in lines 17 through 24 where they are each define in the normal way, except that the class name is added with the function name and separated from it by a double colon (`::`), called **scope operator** as shown below:

```
void item::set(int enter_value)
{
    keep_data = enter_value;
}

int item::get_value(void)
{
    return keep_data;
}
```

- You can do anything with the private data within the function implementations which are a part of that class; also the private data of other classes is hidden and not available within the member functions of this class.
- This is the reason we must mention the class name to the function names of this class when defining them
- It is legal to declare variables and functions in the private part, and additional variables and functions in the public part also.
- In most practical situations, variables only declared in the private section and functions only declared in the public part of a class definition. Occasionally, variables or functions are declared in the other part.

```
#include <iostream.h>

// the class declaration part

class item

{
```

```
    int  keep_data;    // private by default, it is public in struct
   public:              // public part
   void  set(int enter_value);
   int  get_value(void);
};
  // class implementation part
  void  item::set(int enter_value)
{
   keep_data = enter_value;
}
int  item::get_value(void)
{
   return  keep_data;
 }

// main program
void main()
 {
      item    John_cat,  Joe_cat,  Big_cat;
      // three objects instantiated
      int   garfield;    // a normal variable
     John_cat.set(10);   // assigning values
      Joe_cat.set(11);
      Big_cat.set(12);
      garfield = 13;
```

```
        cout<<"Accessing data using class\n";
        cout<<"-------------------------\n";
        cout<<"Data value for John_cat is "<<John_cat.get_value()<<"\n";
        cout<<"Data value for Joe_cat is "<<Joe_cat.get_value()<<"\n";
        cout<<"Data value for Big_cat is "<<Big_cat.get_value()<<"\n";
        cout<<"\nAccessing data normally\n";
        cout<<"--------------------------\n";
        cout<<"Data value for garfield is "<<garfield<<"\n";

          }
```

Output:

```
Accessing data using class
-------------------------
Data value for John_cat is 10
Data value for Joe_cat is  11
Data value for Big_cat is 12
Accessing data normally
--------------------------
Data value for garfield is 13
```

```
// program classobj.cpp - using class instead of struct
   #include <iostream.h>
  // a simple class declaration part
```

```
  class  rectangle
  {
    // private by default, member variables
    int height;
   int width;
  public:
  // public, with two methods
   int area(void);
   void initialize(int, int);
};
 // class implementation part
int rectangle::area(void)
{
    return (height * width);
}
 void  rectangle::initialize(int initial_height, int initial_width)
{      height = initial_height;
     width = initial_width;
}
 // a normal structure - compare it with class
struct  pole
{
int  length;  // public
```

```
    int  depth;   // public

  };
```

Learn about c , what is c , cprogramming , programming c , what c , what is programming , programming, c programming , c programming tutorial ,